포기를 통해, 포기하지 않기

정승현

2022년 9월 16일 금요일

Abstract

무언가를 탐구하다 보면 해결하기 힘든 문제가 있다. 최악의 경우 그 해결하기 힘든 문제를 해결할 수 없음이 증명되기도 한다. 수학에서는 수 체계가 완전하거나, 스스로 무모순함(일관적임)을 증명할 수 있거나, 결정 가능하지 않다는 것이 밝혀졌고, 전산학에서는 유의미한 실행 성질(semantic property)을 유한한 시간 안에 정확히 판별하는 알고리즘이 없다는 것이 증명되었다. 하지만 그것은 여러 방법으로 극복될 수 있다. 수학자들은 더 상위 체계를 활용해 수 체계의 무모순을 증명하거나, 불완전성 정리가 적용된 사례를 확인하고 연구를 이어나갔고, 전산학자들은 전문가의 수동적인 노력을 붓거나, 종료를 보장하지 않거나, 정확한 분석을 보장하지 않음으로서 유용한 도구를 만들어나가고 있고 그 방법에 대해서도 많은 연구가 진행되고 있다. 이렇게 포기를 통해, 포기하지 않을 수 있고, 그렇게 하면 궁극적으로 원하는 목적을 다른 방법으로 천천히 이룰 수 있을 것이다.

'Breakthrough', 돌파구를 뜻하는 영어 단어인데, 무언가 잘 해결되지 않던 것을 극복한 연구를 언급하는 글에서 자주 본 단어이다. 그런 논문에서 직접 자신있게 쓰기도 한다. 무언가를 연구하거나 만들다 보면 해결하기 힘든 문제를 마주한다. 그것이 너무나 일반론적이어서 수학자 전체나 전산학자 전체가 집중하게 되기도 한다. 그리고 문제를 해결할 수 없다는 것이 증명되기도 한다. 그럴 때 돌파구를 찾았던 이야기를 해보고자 한다.

수학자들, 그 중에서 형식주의자들은 괴델과 튜링의 증명으로 큰 벽을 마주했다. 페아노 공리계를 포함하는 형식 체계는 완전(complete)할 수 없고, 스스로 모순이 없다(consistent)는 것을 증명할 수 없어 모순이 없는지 알 수도 없고, 그리고 튜링 기계를 통해 명제의 참과 거짓을 결정 가능(decidable)하지도 않다는 것이 증명되었기 때문이다. 완벽한 형식 체계 안에서 모든 참인 명제를 증명하고 진리에 도달한다는 이상이 '증명'을 통해 부서진 것이다. 이것은 형식주의 수학자가 아니었더라도 충격적이었을 것이다. 나도 수학을 공부하면서 처음 논리와 집합, 양화사(quantifier)를 배우고, 증명을 형식적으로 쓰기 시작했을 때 그 이상을 실현할 수 있지 않을까 기대했는데, 아직 실현되지 않은 수준이 아니라 이상에 도달할 수 없음이 증명되었다는 것이 놀라웠던 기억이 난다.

그러나 형식주의 수학자들은 돌파구를 찾아냈다. 게르하르트 겐첸은 무모순성을 증명하고 싶은 형식 체계보다 상위의 체계를 동원해(더 자세히는, 유한한 기계적 절차를 고집하는 대신 초한귀납법을 적용해) 페아노 공리계를 포함하는 형식 체계의 무모순성 증명을 완성했다. 그리고 오히려 연속체 가설과 그 부정 각각이 ZFC 공리계와 무모순이라는 것이 증명되어 이를 공리로 채용한 공리계와 부정을 공리로 채용한 공리계에서 각각 연구가 진행되는 등 흥미로운 사례도 나오게 되었다. 이외에도 그런 사례는 많다. 5차 이상의 다항방정식의 근의 방정식을 만들 수 없다는 것이 증명되었지만, 다항방정식의 근이 정말로 활용되는 경우에는 허용되는 오차 범위가 있다. 그러므로 원하는 범위 이내로 얼마든지 오차를 줄일 수만 있다면 문제가 없고, 교과서에서 배우는 이분법(bisection method)이나 뉴턴 방법(Newton's method)만으로도 이를 달성할 수 있다.

전산학자들도 그랬는데, 튜링의 결정 불가능성(undecidability) 정리의 따름 정리인 라이스의 정리(Rice's theorem)로, 어떠한 자명하지 않은, 즉 유의미한 실행 성질(semantic property)에 대해서도 유한한 시간 안에 정확히 판별할 수 있는 알고리즘이 존재하지 않는다는 것을 알게 되었다. 원하는 방식으로 작동하는 컴퓨터 프로그램을 만들고(프로그램 합성; program systhesis), 컴퓨터 프로그램에 오류가 없는지 검사하고(프로그램 검증; program verification), 프로그램이 원하는 성질을 가지는지, 혹은 원하지 않는 오류가 없는지 분석(프로그램 분석, program analysis)하여 프로그래밍 시스템을 '아름답게' 만들고 싶었을텐데, 많은 것의 자동화를 기대받는 좋은 컴퓨터가 개발되기도 전에 일찍이 그것을 완벽히 수행하는 알고리즘을 만들 수 없다는 것이 증명된 것이다.

하지만 전산학자들도 포기를 통해, 포기하지 않았다. 자동으로 분석하는 것(automatic)을 포기하거나, 분석이 완료되는 것을 보장하는 것(terminating)을 포기하거나, 정확히 분석하는 것(exact)을 포기함으로써 그렇게 상황에 맞게 분석하는 방법을 연구하고 구현체를 만들면서 프로그래밍 시스템을 점점 '아름답게'

만들고 있다. 특히 타입 검사와 같은 안전(sound)한 정적 분석(static analysis), 퍼징과 같은 완전(complete)한 동적 분석(dynamic analysis)은 많이 발전해서, 컴파일러에 기본적으로 포함되어 오류를 줄여주고 있고, 나와 학우들이 듣는 과목들의 프로그래밍 과제 채점에도 활용되고 있다.

나는 공학자들, 과학자들, 수학자들의 이런 포기하면서도 포기하지 않는 태도가 아름답다고 생각하고, 그렇게만 한다면 궁극적으로 바라던 목적은 결국 이룰 수 있을 거라고 생각한다. 마주친 커다란 벽 너머에 어떤 로망 내지는 꿈을 가지고 있던 사람들이 무너지는 때가 있는데, 절대로 부술 수도 우회할 수도 없다는 것을 누군가 알아냈을 때이다. 사람들은 그 꿈을 포기해야 해서 절망하거나 로망을 포기해야 해서 아쉽겠지만, 사실은 원래 목적까지 포기하지는 않아도 될 때가 많다. 돌파구(breakthrough)라는 단어는 정말 그 벽을 부수어(break) 뚫는 느낌을 주지만, 가끔은 가로막은 그것이 XY문제의 Y라는 것을 알아채는 것이기도 하다. 그 벽을 부술 수 없다고 다른 것들이 의미가 없게 되지 않기 때문이다. 가령 그 벽 너머에 있을 것 같은 아름다운 노랫소리를 원했던 거라면, 벽에 귀를 대고 들어볼 수도 있겠다.